محمد امانلو تکلیف هشتم درس پایگاه داده، استاد شاکری 810100084

سوال اول) الف

استفاده از متد IO-Chunked به کاهش هزینه ورودی/خروجی (IO) کمک می‌کند زیرا در این روش صفحات به صورت بلوکی و پشت سر هم خوانده می‌شوند. این روش در مقایسه با حالتی که صفحات پراکنده و در مکان‌های مختلف دیسک قرار دارند و برای هر صفحه یک عملیات IO جداگانه انجام می‌شود، کارایی بهتری دارد و منجر به کاهش زمان و هزینه IO می‌شود. اما در این میان یک معاوضه یا trade-off مهم بین اندازه بلوک‌ها و هزینه وجود دارد. با افزایش اندازه بلوک‌ها، تعداد passهای مورد نیاز برای پردازش داده‌ها بیشتر می‌شود. این به این دلیل است که با افزایش اندازه بلوک‌ها، out-fan یا تعداد بلوک‌هایی که می‌توانند در هر مرحله پردازش شوند، کاهش می‌یابد. بنابراین، نیاز به پاس‌های بیشتری برای تکمیل پردازش کل داده‌ها است که به نوبه خود می‌تواند هزینه IO را افزایش دهد. در صورتی که اندازه بلوک‌ها بیش از حد بزرگ باشد، افزایش تعداد passها می‌تواند تاثیر منفی بیشتری روی هزینه IO بگذارد و در نهایت منجر به افزایش کلی هزینه‌ها شود. بنابراین، باید یک تعادل مناسب بین اندازه بلوک‌ها و تعداد passها برقرار شود تا هزینه IO بهینه شود. به طور کلی، یک معاوضه بین افزایش اندازه بلوک‌ها (که منجر به کاهش هزینه‌های IO می‌شود) و افزایش تعداد passها (که می‌تواند هزینه‌های IO را افزایش دهد) وجود دارد. انتخاب اندازه بلوک مناسب به گونه‌ای که این تعادل به بهترین شکل ممکن حفظ شود، کلید کاهش هزینه IO در متد IO-Chunked است.

ب) بخش هزینه‌بر عملگر projection زمانی است که نیاز به حذف تاپل‌های تکراری (duplicate tuples) باشد، که با استفاده از واژه DISTINCT مشخص می‌شود. در این حالت، برای حذف تاپل‌های تکراری دو روش اصلی وجود دارد:

1. روش Sorting: در این روش، ابتدا فیلدهایی که قرار است انتخاب شوند، مرتب‌سازی (sort) می‌شوند. پس از مرتب‌سازی، تاپل‌های تکراری که در کنار یکدیگر قرار می‌گیرند به راحتی قابل شناسایی و حذف هستند. مزیت این روش در سادگی و کارآمدی آن در حذف تاپل‌های تکراری پس از مرتب‌سازی است، زیرا تاپل‌های مشابه در کنار هم قرار دارند.

2. روش Hashing: در این روش، روی فیلدهایی که قرار است انتخاب شوند، عملیات هشینگ انجام می‌شود. پارتیشن‌های حاصل از هشینگ به حافظه منتقل می‌شوند و با ساختن ساختار هش در حافظه، تاپل‌های تکراری شناسایی و حذف می‌شوند. مزیت این روش در عملکرد سریع آن برای مجموعه داده‌های بزرگ است، به خصوص زمانی که داده‌ها به طور یکنواخت در پارتیشن‌های مختلف توزیع شده‌اند.
توضیح مفصل‌تر:

- روش Sorting: این روش زمانی کارا است که تعداد تاپل‌ها متوسط باشد و فضای کافی برای مرتب‌سازی در دسترس باشد. با استفاده از الگوریتم‌های مرتب‌سازی کارا مثل Merge Sort یا Quick Sort، فیلدهای مورد نظر مرتب شده و پس از آن تاپل‌های تکراری حذف می‌شوند. اگر حجم داده‌ها بزرگ باشد، مرتب‌سازی می‌تواند زمان‌بر و پرهزینه شود، اما با توجه به اینکه تاپل‌های تکراری در کنار هم قرار می‌گیرند، فرآیند حذف آنها ساده‌تر و سریع‌تر است.

- روش Hashing: این روش مناسب‌تر برای داده‌های حجیم است، زیرا هشینگ می‌تواند به صورت کارا داده‌ها را به پارتیشن‌های کوچک‌تر تقسیم کند. هر پارتیشن به صورت جداگانه در حافظه پردازش می‌شود و با استفاده از ساختارهای هش، تاپل‌های تکراری شناسایی و حذف می‌شوند. این روش نیازمند فضای حافظه کافی برای نگهداری ساختارهای هش است و در صورتی که حافظه کافی نباشد، ممکن است نیاز به چند پاس داشته باشد که هزینه‌ها را افزایش می‌دهد.

در نهایت، انتخاب روش مناسب بستگی به اندازه داده‌ها، میزان تکراری بودن تاپل‌ها و منابع موجود (مانند حافظه و زمان) دارد. هر کدام از این روش‌ها در شرایط خاص خود می‌توانند کارآمد باشند و هدف اصلی، کاهش هزینه‌های پردازشی در حذف تاپل‌های تکراری است.

ج) در یک پلن fully piped line نمی‌توان از روش sort merge join استفاده کرد. دلیل این امر به نحوه عملکرد هر دو روش برمی‌گردد:
پلن fully piped line: در این پلن، داده‌ها به صورت پیوسته و بدون توقف از یک مرحله به مرحله بعدی منتقل می‌شوند. هیچ واسطه‌ای برای ذخیره‌سازی موقت داده‌ها وجود ندارد. هدف اصلی این پلن کاهش زمان تأخیر و افزایش بهره‌وری با استفاده از انتقال پیوسته داده‌ها بین مراحل مختلف پردازش است.

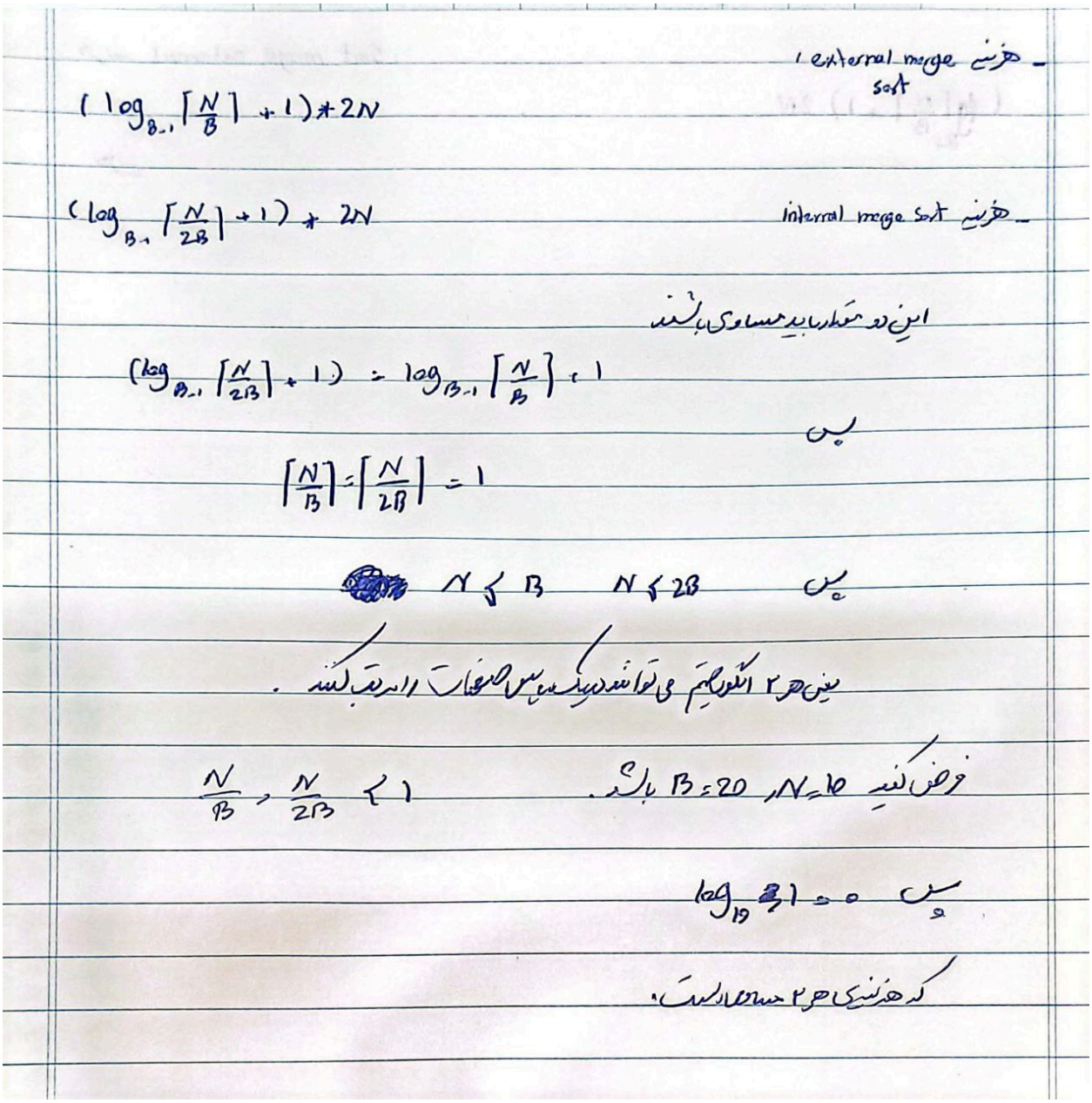
روش sort merge join: این روش نیازمند مرتب‌سازی هر دو مجموعه داده‌ای است که قرار است با هم join شوند. بعد از مرتب‌سازی، داده‌ها باید به صورت موقت ذخیره شوند تا امکان انجام عملیات merge وجود داشته باشد. در حین انجام عملیات merge، نتایج میانی باید ذخیره و مجدداً مرتب‌سازی شوند تا نتیجه نهایی به دست آید. این فرآیند نیاز به فضای ذخیره‌سازی موقت برای نگهداری نتایج میانی دارد.

دلیل عدم استفاده از sort merge join در پلن fully piped line
1. نیاز به ذخیره‌سازی موقت: روشsort merge join نیازمند نوشتن و خواندن موقت داده‌ها در دیسک است تا مرتب‌سازی و merge انجام شود. در پلن fully piped line، هیچ مرحله‌ای برای ذخیره‌سازی موقت داده‌ها وجود ندارد.
2. ماهیت پیوسته پردازش: پلنfully piped line طراحی شده است تا داده‌ها را به صورت پیوسته پردازش کند، بدون توقف برای ذخیره‌سازی یا مرتب‌سازی. این با نیازهای sort merge join که مستلزم توقف برای ذخیره و مرتب‌سازی است، سازگار نیست.
3. کارایی و بهره‌وری: هدف اصلی پلن fully piped line کاهش تأخیر و افزایش بهره‌وری

از طریق پردازش پیوسته است. نیاز به مرتب‌سازی و ذخیره‌سازی موقت در sort merge join با این هدف در تضاد است و باعث افزایش زمان پردازش می‌شود.

استفاده از sort merge join در یک پلن fully piped line به دلیل نیاز به ذخیره‌سازی موقت و مرتب‌سازی میانی داده‌ها امکان‌پذیر نیست. در پلنfully piped line، داده‌ها به صورت پیوسته پردازش می‌شوند و هیچ مرحله‌ای برای نوشتن و خواندن موقت داده‌ها وجود ندارد، در حالی که sort merge join به این فرآیندها نیازمند است. بنابراین، این دو روش با یکدیگر سازگار نیستند و نمی‌توانند به طور همزمان در یک پلن fully piped lineمورد استفاده قرار گیرند.

د) برای پیدا کردن سناریویی که هزینه sort merge external با sort merge internal یکی باشد، باید شرایطی را بررسی کنیم که در آن هزینه هر دو برابر باشد. برای این کار، از فرمول‌های داده شده برای هزینه‌ها استفاده می‌کنیم:



سوال دوم) 1. B-tree: این نوع شاخص برای عملیات‌های مقایسه‌ای مانند <, >, =<, =>, >= مناسب است. کوئری شامل شرط $a > 9$ و $b < 8$ است که هر دو عملیات مقایسه‌ای هستند. چون شروط a و b به ترتیب prefix شاخص هستند، می‌توان از B-tree استفاده کرد. در مقابل شاخص‌های Hash تنها برای عملیات‌های برابر (=) مناسب هستند. این کوئری شامل شرایط مقایسه‌ای است و نمی‌توان از شاخص Hash استفاده کرد.

2. B-tree: ابتدا کوئری را به دو بخش تقسیم می‌کنیم: (a = 1 AND b < 7) و (a = 1 AND c = 2). برای شرط (a = 1 AND b < 7)، شاخص B-tree قابل استفاده است چون a و b به ترتیب prefix شاخص هستند. برای شرط (a = 1 AND c = 2)، چون c در prefix شاخص بعد از a قرار دارد، نمی‌توان به طور مستقیم از B-tree استفاده کرد. در مقابل این کوئری شامل عملیات OR است که برای شاخص‌های Hash مناسب نیستند. همچنین، برای استفاده از شاخص Hash، تمامی کلیدها باید مقدار دقیق داشته باشند که در اینجا وجود ندارد. پس نمی‌توان به طور کامل از هیچ یک از شاخص‌ها استفاده کرد. اما می‌توان از شاخص B-tree برای حذف رکوردهای غیرمرتبط با شرط a = 1 و سپس انجام اسکن بر روی رکوردهای باقی‌مانده استفاده کرد.

3. مثل مثال قبلی نمی توان از هیچ شاخصی استفاده کرد اما B-tree: ابتدا کوئری را به دو بخش تقسیم می‌کنیم: (a = 1 OR b < 7) و (a = 1 OR c = 2). برای هر دو بخش کوئری، نمی‌توان به طور مستقیم از شاخص B-tree استفاده کرد زیرا شرایط OR در عملیات جستجو مناسب نیستند. همچنین در Hash: این کوئری شامل شرایط OR است که برای شاخص‌های Hash مناسب نیستند. همچنین، نیاز به مقدار دقیق تمامی کلیدها برای استفاده از شاخص Hash وجود دارد که در اینجا وجود ندارد. پس نمی‌توان از هیچ یک از شاخص‌ها استفاده کرد. اگرچه برای بخش‌هایی از کوئری می‌توان از B-tree استفاده کرد، اما به دلیل وجود شرط OR، این امکان به طور کامل فراهم نیست.

سوال سوم) برای تحلیل سناریوهای مختلف در استفاده از شاخص‌ها (index) و تعیین بهترین روش دسترسی، باید هزینه I/O هر روش را محاسبه کنیم. برای این کار، نیاز به محاسبه تعداد صفحات دیسکی است که باید خوانده شوند. داده‌های فرض شده به صورت زیر هستند:

- تعداد کل صفحات: 100
- تعداد کل رکوردها: 20,000
- ضریب کاهش (reduction factor) هر شرط: 0.1

شرط‌های کوئری:

1- sid = 102

2- bid = 6

3- day > 8/9/98

شاخص‌های موجود:

- B-tree روی day (ممکن است clustered یا unclustered باشد)

- Hash روی (sid, bid) (ممکن است clustered یا unclustered باشد)

سناریو 1: شاخص Hash clustered و B-tree unclustered

هزینه استفاده از شاخص Hash clustered:

$$0.01 = 0.1 \times 0.1 = \text{Reduction factor}$$

$$1 = 100 \times 0.01 = \text{تعداد صفحات خوانده شده}$$

- هزینه استفاده از شاخص B-tree unclustered:

$$0.1 = \text{Reduction factor}$$

$$2000 = 20000 \times 0.1 = \text{تعداد رکوردهای مورد بررسی}$$

$$10 = 2000/200 \approx (\text{records per page})/2000 = \text{تعداد صفحات خوانده شده}$$

- هزینه اسکن کل فایل‌ها:

$$100 = \text{تعداد صفحات خوانده شده}$$

در این سناریو، شاخص Hash clustered با هزینه 1 صفحه بهترین عملکرد را دارد.

سناریو 2: شاخص Hash unclustered و B-tree clustered

- هزینه استفاده از شاخص Hash unclustered:

$$0.01 = 0.1 \times 0.1 = \text{Reduction factor}$$

$$200 = 20000 \times 0.01 = \text{تعداد رکوردهای مورد بررسی}$$

$$1 = 200/200 = (\text{records per page})/200 = \text{تعداد صفحات خوانده شده}$$

اما در هر صورت هر رکورد باید به طور جداگانه خوانده شود، بنابراین:

تعداد صفحات خوانده شده = 200

هزینه استفاده از شاخص B-tree clustered:

$$0.1 = \text{Reduction factor}$$

$$10 = 100 \times 0.1 = \text{تعداد صفحات خوانده شده}$$

هزینه اسکن کل فایل‌ها:

تعداد صفحات خوانده شده = 100

نتیجه: در این سناریو، شاخص B-tree clustered با هزینه 10 صفحه بهترین عملکرد را دارد.

سناریو 3: هر دو شاخص unclustered

هزینه استفاده از شاخص Hash unclustered:

$$0.01 = 0.1 \times 0.1 = \text{Reduction factor}$$

$$200 = 20000 \times 0.01 = \text{تعداد رکوردهای مورد بررسی}$$

$$200 = \text{تعداد صفحات خوانده شده}$$

هزینه استفاده از شاخص B-tree unclustered:

$$0.1 = \text{Reduction factor}$$

$$2000 = 20000 \times 0.1 = \text{تعداد رکوردهای مورد بررسی}$$

$$10 = 2000/200 \approx (\text{records per page})/2000 = \text{تعداد صفحات خوانده شده}$$

هزینه اسکن کل فایل‌ها:

تعداد صفحات خوانده شده = 100

نتیجه: در این سناریو، اسکن کل فایل‌ها با هزینه 100 صفحه بهترین عملکرد را دارد.

ب) بهینه‌ساز کوئری (Query Optimizer) برای تعیین بهترین روش دسترسی به داده‌ها و شناسایی سناریوهای مختلف از اطلاعات موجود در سیستم کاتالوگ (System Catalog) استفاده می‌کند. در این بخش، نحوه استفاده بهینه‌ساز کوئری از کاتالوگ سیستم و اطلاعات موجود در آن برای شناسایی سناریوها و انتخاب روش بهینه را توضیح می‌دهیم.

System Catalog شامل مجموعه‌ای از جداول و داده‌هایی است که اطلاعاتی درباره ساختار و ویژگی‌های دیتابیس و شاخص‌ها فراهم می‌کنند. برخی از اطلاعات موجود در System Catalog شامل موارد زیر هستند:

1. اطلاعات جداول Tables Information
2. اطلاعات شاخص‌ها (:Indexes Information)
3.اطلاعات مربوط به توزیع داده‌ها (Statistics Information):

نحوه استفاده بهینه‌ساز کوئری از System Catalog

بهینه‌ساز کوئری با استفاده از اطلاعات موجود در System Catalog به صورت زیر عمل می‌کند:

1. بررسی شاخص‌های موجود: بهینه‌ساز بررسی می‌کند که چه شاخص‌هایی روی ستون‌های مورد استفاده در کوئری وجود دارند. این اطلاعات شامل نوع شاخص (hash یا B-tree) و نوع آن (clustered یا unclustered) است.

2. محاسبه هزینه‌ها: بهینه‌ساز با استفاده از اطلاعات کاتالوگ، هزینه دسترسی به داده‌ها را برای هر شاخص محاسبه می‌کند. این محاسبات شامل تعداد صفحات خوانده شده و هزینه‌های I/O می‌باشد.

3. انتخاب سناریو بهینه: بهینه‌ساز با مقایسه هزینه‌های محاسبه شده برای هر روش دسترسی (استفاده از شاخص Hash، شاخص B-tree یا اسکن کل فایل‌ها)، بهترین روش را انتخاب می‌کند.

بهینه‌ساز کوئری با استفاده از اطلاعات موجود در System Catalog (مانند تعداد رکوردها، تعداد صفحات، نوع شاخص‌ها و توزیع داده‌ها) قادر است بهترین روش دسترسی به داده‌ها را انتخاب کند. این اطلاعات بهینه‌ساز را در شناسایی سناریوهای مختلف و انتخاب روش بهینه برای اجرای کوئری یاری می‌دهد.

سوال چهارم)

الف) Page-Oriented Nested Loop Join

1. اگر Enrollments به عنوان outer انتخاب شود:

$$\text{هزینه} = (300 \times 600) + 600 = 180600$$

2. اگر Students به عنوان outer انتخاب شود:

$$\text{هزینه} = (600 \times 300) + 300 = 180300$$

Index Nested Loop Join

1. اگر Enrollments به عنوان inner انتخاب شود:

$$\text{هزینه} = ((10 + 1.2) \times 20 \times 300) + 300 = 67500$$

در این محاسبه فرض شده که هزینه پیدا کردن رکوردهای منطبق در جدول Students با

استفاده از شاخص B-tree clustered برابر با (1.2 + 10) است (1.2 برای جستجوی شاخص و 10 برای تعداد کورس‌های هر دانشجو).

2. اگر Students به عنوان inner انتخاب شود:

$$\text{هزینه} = ((1+3) \times 150 \times 600) + 600 = 360600$$

در این محاسبه فرض شده که هزینه پیدا کردن رکوردهای منطبق در جدول Enrollments با استفاده از شاخص Hash unclustered برابر با (3 + 1) است (3 برای جستجوی شاخص و 1 برای تعداد رکوردهای هر دانشجو).

ب) فرمول هزینه external merge sort:

$$\left(\log_{\lceil \frac{N}{B} \rceil}^{B-1} + 1\right) \times 2N$$

برای جدول Enrollments داریم که : N= 600 , B = 5

$$\text{هزینه} = \left(\log_{4}^{\lceil \frac{600}{5} \rceil} + 1\right) \times 600 \times 2 = 6000$$

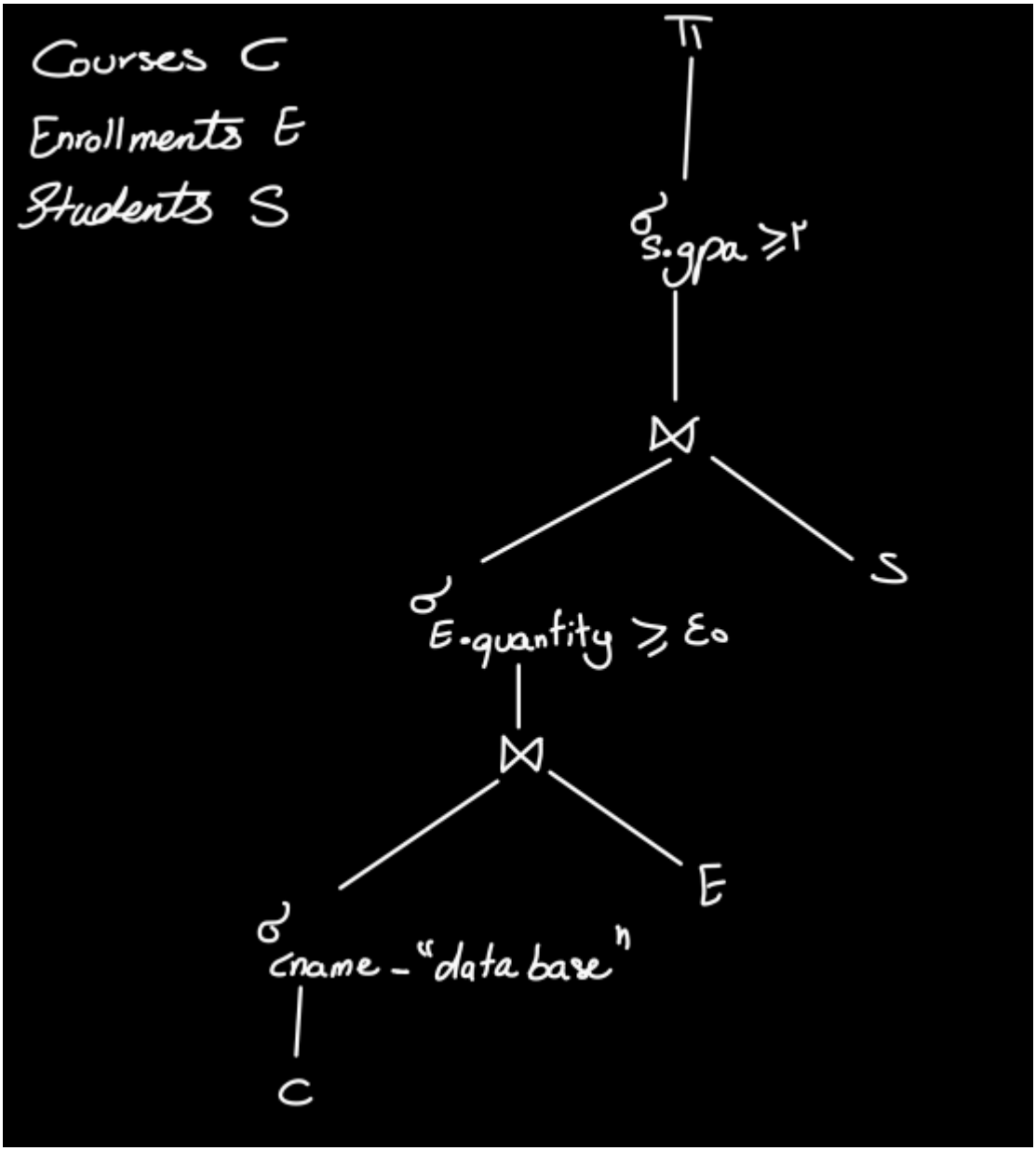
چون که یک شاخص clustered بر روی جدول Students تعریف شده نیازی به سورت کردن آن نیست و هزینه نهایی بخش sort برابر هزینه سورت کردن Enrollments یعنی 6000 است.

سوال پنجم)

```
Select S.sid, C.cid

From Students S, Enrollments E, Courses C
```

Where S.sid = E.sid AND C.cid = E.cid AND S.gpa >= 2 AND E.quantity > 40 AND C.cname = 'Database



مراحل بهینه‌سازی و محاسبه هزینه‌ها:

1. ابتدا عمل انتخاب بر روی جدول `Courses` انجام می‌شود. این انتخاب با استفاده از شاخص `tree unclustered` که بر روی فیلد `cname` تعریف شده،

انجام می‌شود. دلیل شروع با این انتخاب این است که بعد از اعمال این فیلتر، تعداد تاپل‌ها به 20 کاهش می‌یابد، که در هیچ فیلتر دیگری اینطور نیست.

$$\text{هزینه} = \text{عبور از درخت} + \text{خواندن صفحات} = 3 + 20 = 23$$

2. در این مرحله، join بین تاپل‌های انتخاب شده از `Courses` و جدول `Enrollments` انجام می‌شود. از آنجا که بر روی فیلد `cid` در جدول `Enrollments` شاخص تعریف شده است، از روش INLJ استفاده می‌شود و داده‌های انتخاب شده از `Courses` به عنوان outer در نظر گرفته می‌شوند. تعداد تاپل‌های بعد از join برابر با 200 خواهد بود (به ازای هر درس 10 ثبت‌نام داریم).

$$\text{هزینه} = \#\text{تاپل‌ها} \times (\text{عبور از درخت} + \text{خواندن صفحات}) = 20 \times (10 + 3) = 260$$

3. انتخاب بر روی فیلد `quantity` به صورت on-the-fly انجام می‌شود. با توجه به توزیع یکنواخت `quantity` بین 10 تا 60، انتظار می‌رود 40 درصد تاپل‌ها باقی بمانند، یعنی 80 تاپل.

$$\text{هزینه} = \#\text{تاپل‌ها} \times (\text{جستجوی باکت hash} + \text{خواندن صفحات}) = 80 \times (1 + 1.2) = 176$$

4. در این مرحله، join بین تاپل‌های باقی‌مانده و جدول `Students` انجام می‌شود. در اینجا هم از روش INLJ استفاده می‌شود چون بر روی فیلد `sid` جدول `Students` شاخص داریم. خروجی انتخاب قبلی به عنوان outer در نظر گرفته می‌شود و به ازای هر تاپل در outer یک تاپل متناظر `Students` وجود دارد، یعنی 80 تاپل باقی می‌مانند.

5. انتخاب بر روی فیلد `gpa` به صورت on-the-fly بر روی خروجی مرحله قبل انجام می‌شود. با توجه به هیستوگرام، 60 درصد داده‌ها gpa بزرگتر مساوی 2 دارند، پس خروجی نهایی 48 تاپل خواهد بود.
6. در نهایت، projection انجام می‌شود.

$$\text{هزینه کل} = 459 = 176 + 260 + 23$$

سوال ششم) الف)

- N = 10000 (تعداد صفحات)

- B = 8 (تعداد صفحات بافر)

تعداد پاس‌های مورد نیاز:

$$\text{تعداد پاس‌ها} = \left(\frac{N}{2B}\right)_{B-1}\log + 1 = \left(\frac{10000}{16}\right)_{7}\log + 1$$

$$4.1 \approx \log_7(625)$$

پس تعداد پاس‌ها:

5 = 4 + 1

هزینه I/O:

$$\text{هزینه} = 100000 = 5 \times 10000 \times 2 = (\left(\frac{N}{2B}\right)_{B-1}\log + 1) \times 2N$$

ب) تعداد و اندازه runهای تولید شده در هر پاس:

- Pass 0:

$$\text{تعداد runها} = \left\lceil \frac{10000}{16} \right\rceil = 625$$

هر run طولش 16 صفحه است.

- Pass 1:

$$\text{تعداد runها} = \left\lceil \frac{625}{7} \right\rceil = 90$$

هر run طولش 112 صفحه است به جز آخرین run که طولش 32 صفحه است.

- Pass 2:

$$\text{تعداد runها} = \left\lceil \frac{90}{7} \right\rceil = 13$$

هر run طولش 784 صفحه است به جز آخرین run که طولش 592 صفحه است.

- Pass 3:

$$\text{تعداد runها} = \left\lceil \frac{13}{7} \right\rceil = 2$$

هر run طولش 5488 صفحه است به جز آخرین run که طولش 4512 صفحه است.

- Pass 4:

$$\text{تعداد runها} = \left\lceil \frac{2}{7} \right\rceil = 1$$

یک run با طول 10000 صفحه داریم.

ج) استفاده از روش IO-Chunked برای کاهش هزینه I/O:

برای ثابت نگه داشتن تعداد پاس‌ها، سایز بلوک باید به نحوی انتخاب شود که تعداد پاس‌ها تغییری نکند.

$$1 - \left\lfloor \frac{B}{b} \right\rfloor = F$$

تعداد پاس‌ها:

$$5 = \left\lceil \left(\frac{N}{2B} \right)_F \log \right\rceil + 1$$

$$4 \approx \log_F(625)$$

پس: $\log_F(625)$ بین 3 و 4 است. ($4 \geq \log_F(625) > 3$) و تقریبا برابر 4

در نتیجه می توان نوشت:

$$\frac{8}{6} \geq b > \frac{8}{10} \Rightarrow 9 \geq \left\lfloor \frac{8}{b} \right\rfloor \geq 6 \Rightarrow 8.3 > 1 - \left\lfloor \frac{8}{b} \right\rfloor \geq 5$$

بنابراین با قرار دادن مقدار 1 در b می‌توان تعداد پاس‌ها را ثابت نگه داشت.